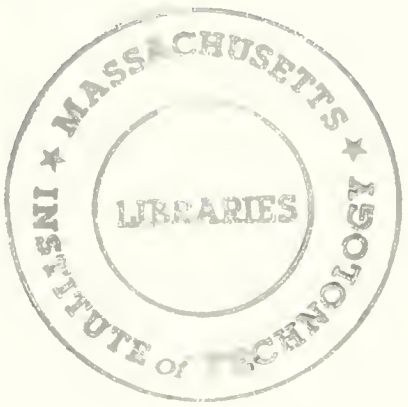
MIT LIBRARIES
3 9080 00699020 1


MASSACHUSETTS
INSTITUTE of TECHNOLOGY
LIBRARIES

DEWEY

WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

The Changing Role of the User in the Development of Application Software

Bradley A. Feld

WP# 3152-90-BPS August 1990

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139

OCT 23 1990

The Changing Role of the User in the Development of Application Software

Bradley A. Feld

WP# 3152-90-BPS August 1990

M.I.T. LIBRARIES
OCT 2 3 1990
RECEIVED

Abstract

In seeking to better understand the sources of innovation, researchers have investigated the division of labor between user and manufacturer in the development of new products. Specifically, the "partitioning of tasks" between user and manufacturer has drawn attention.

In this paper, I explore the changing role of the user in the development of application software. I argue that responsibility for application development is shifting from generic manufacturers of software to the users of the software. I hypothesize that, given this shift, an efficient division of labor entails manufacturers who develop "tools" and transfer them to the users. The user then develops his or her own applications using the tools developed by the manufacturer.

I carried out a pilot study on the actual partitioning of tasks between the user of application software and the specialist developer of the software (the manufacturer). In this study, I investigated several software projects undertaken by a single custom software firm for three clients seeking application software to automate aspects of their businesses. Finally, I discuss several trends in software technology that could arise from this ongoing shift of responsibility to the user.

The Changing Role of the User in the Development of Application Software

1 Introduction[1]

Although most software applications are currently developed by programmers, innovations in software programming tools and user-friendly methods to link these tools increasingly provide users who are not programmers with the ability to develop their own applications more efficiently and cost effectively. These innovations are in the process of transforming the way businesses and individuals approach application software development.

Consider the development of a financial model for forecasting cash flow. A decade ago, the user was required to specify the problem to a programmer, who would then assume the task of writing the program in a complex programming language such as COBOL or BASIC. If the user needed any changes made after he began using the program, he would have to interrupt his work and try to describe his needs to the programmer, who would then take the necessary steps to incorporate the changes into the program. Today, a user can implement her financial model directly in a spreadsheet program. She controls the entire development cycle according to her specific needs and can quickly and freely make any necessary changes to the program as she works.

The current evolution of software development tools appears to be shifting the responsibility for application development from a specialist developer of software (the programmer) to the user of the software. New programming environments for developing application software (such as fourth-generation and object-oriented languages) are appearing that not only dramatically decrease the cost of developing application software

1. I would like to thank Eric von Hippel and David Jilk for their stimulating and insightful contributions to the work contained in this paper.

and increase the value of the software to the user, but also enable the user to develop his or her own applications. The impact of this trend will, I believe, lead to a new, more effective division of labor between the developer and the user of software.

In this paper, I explore the changing role of the user in the development of application software and seek to establish an empirical basis for research on the actual partitioning of application programming tasks between the user and the manufacturer and on the implications for improved practice. I first summarize some literature that exists concerning this subject (section 2). I then turn to a brief history of the roles of the end user and specialist programmer -- often employed by a software "manufacturing" firm -- in the overall development of application software (section 3). Next, I discuss a pilot study that I conducted to explore the different ways in which software development tasks are partitioned between users and programmers of software and the efficiency with which information that affects application development is transferred between user and programmer (section 4). In this study, I investigated three "first-of-kind" projects that were developed for new clients by a custom software company.[2] I examined various qualities of the information passed back and forth across organizational boundaries such as the initiator of the request for change (user or manufacturer), the granularity (chunks vs. items) of the messages transferred across organizational boundaries, the completeness or incompleteness of messages transferred, and the content of users' requests (new features vs. bugs). Finally, I discuss some trends I have observed that will increasingly have an impact on the role of the user in the development of application software (section 5).

---

2. I define application software as software developed for a specific user need. This is often referred to as "custom software."

2 The Literature

Various researchers have examined the division of labor between the user and manufacturer in the innovation process. Von Hippel explored the relationship between user and manufacturer by examining the sources of innovation (von Hippel 1988) and the implications of task partitioning (von Hippel 1989). He found that problem-solving interdependencies among tasks can be predicted and then managed by either adjusting the boundaries between tasks or reducing the barrier to problem-solving interactions across boundaries. More recently, he discussed the effects of what he terms "sticky data" on the locus of problem-solving and innovation (von Hippel 1990). He found that much of the data needed for problem-solving has "sticky" qualities that make them difficult or too costly to move to other sites. He hypothesized that "if data needed by a problem-solver are sticky, related problem-solving activity must move to and among loci of sticky data -- perhaps repeatedly as problem-solving proceeds" (von Hippel 1990). Von Hippel proposes that "it is an economy, other things being equal, to design problems so that problem-solving draws on only one locus of sticky data..." (von Hippel 1990).

A number of other researchers have investigated how the software development process is organized and in what ways and how effectively it incorporates user and manufacturer input. These studies have been concerned primarily with partitioning of tasks in technical organizations. Salaway (1987) suggests that interactions among members of a team occur at the intersection of user knowledge and designer knowledge. Henderson and Lee (1989) found that during the development of an information system, designers (software developers) were expected to build the system, but domain representatives (users) were expected to supply relevant business knowledge and user input to the designers throughout the process. Finally, Orlikowski (1989) examined the changing

relationship between user and developer in the development of information systems and observed a shift in the division of labor between functional developers and technical developers in a custom software organization following the introduction of computer-aided software engineering (CASE) tools. All of these examples indicate that there is a substantial transfer of information between software user and software developer during the development of application software.

3 Roles of the Programmer and the End User in the Development of Software

The software industry is being transformed by an increasing separation of programming tasks: The creation of application-generating programs (a "tool") and the development of application software (an "application") are no longer a single task controlled by the programmer, as they once were.

In the 1960's, programmers using languages such as Assembly handled all software tasks. Even when the distinction between a "tool" and an "application" did exist, the skills a programmer needed to build a tool were the same skills needed to manipulate the tool to create an application.

Over time, programmers realized that their work could be facilitated by the design of reusable, self-contained modules. These modules, or tools, became fundamental components of newer programming languages. While early programming languages such as Assembly contained no tools, later programming languages such as COBOL included some built-in, although limited, tools for developing input screens and reports. Current programming languages (such as Focus, Dataflex, or Clarion[3]) contain numerous tools but

3. Focus, Dataflex, and Clarion are products of Information Builders, Data Access Corporation, and Clarion Software Company, respectively.

have added simple "user-friendly" methods to link these modules. For example, a programmer using an "object-oriented"[4] programming language such as Actor[5] can create a "window" on the computer screen that behaves like a text editor. A decade ago, this operation would have taken thousands of lines of programming code written by an expert programmer; today, it takes one line of programming code written by a user.

At the same time, advances in the power and speed of computer hardware have permitted software programs to use larger self-contained modules without diminishing the performance of the software. As computing power becomes less expensive, software modules have become more complex without having a visible effect on the performance of the application.

Together with this trend toward increased computing power, trends toward self-contained modules and easy, user-friendly methods to link these modules have automated tasks that previously could only be performed by specialist programmers and have enabled the user increasingly to program his or her own applications. Today, a user can take various self-contained modules originally developed by a specialist programmer and link or reconfigure them by means of "macros"[6] such as those within Lotus 1-2-3.[7] For example, the financial spreadsheet forecasting cash flows that was discussed in the introduction might have been implemented through a Lotus 1-2-3 macro. In this case, the macro (created by the user) only runs within the context of a program (Lotus 1-2-3). Consequently, the program, created by a specialist programmer working for Lotus

---

4. For a good description of object-oriented programming, see the articles by either Wegner (1989) or Meng (1990).

5. Actor is a product of the Whitewater Group.

6. A macro is a "program within a program" comprised of a series of commands or keystrokes specific to a software program that the user can assemble to act as an application to solve a specific need.

7. Lotus 1-2-3 is a spreadsheet program from Lotus Development Corporation.

Corporation, is the application generator while the macro, developed by the user, is the application. The pilot study discussed below seeks to take a first step toward understanding the impact of such shifts.

4 Pilot Study

4.1 Approach

My goal in this study was to understand how application software development is partitioned between the end user and the software developer and how information is transferred -- or not transferred -- between these two during the course of software development. In order to see such transfers clearly, I have elected to study situations in which end users and the specialist programmers attempting to develop software to serve their needs were employed by different firms. Specifically, I studied the sources of 516 improvements in the application software developed for three clients ("users") by a single custom software application company ("software manufacturer").

The custom software firm I elected to study has been in business for five years and has developed over 30 personal computer-based custom software applications for a wide variety of businesses. In the process, the firm also created numerous software development process tools and gained a significant amount of know-how based around a popular fourth-generation language called Clarion.[8] Some sample systems the firm has developed include software for group medical practices, medical laboratory billing, venture capital portfolio management, legal billing, insurance claim processing, real estate investment analysis, rental store point-of-sale, and international accounting.

---

8. Clarion Professional Developer, Clarion Software Company, Pompano Beach, FL.

Because of the way the software development firm approached custom software development, a rich set of contemporaneous data was available for study. Using a methodology similar to rapid prototyping,[9] the custom software company quickly generates an initial, partial solution to the client's problem and develops a custom application. The client starts to use this software and then makes specific requests for changes. The software firm responds and through these interactions the initial solution is greatly elaborated and improved over time through a series of "revisions" to the software issued as software updates to the client. At the start of a client relationship, software revisions may be shipped to the client as often as every two weeks. After a year, the frequency may decline to perhaps one revision every two months.

To ensure that the user and manufacturer databases were distinct from each other at the start of the study, I decided to focus on "first-of-kind" software development projects -- projects in industries in which the custom software firm was working for the first time. In a first-of-kind project, the software firm had no significant knowledge of the client's field of business and the client had not previously been involved in a custom software project. Each of the two databases (user and manufacturer) was, at least at the start of the development project, in a different organization and at a geographically distinct location. The choice to study first-of-kind projects was based on the observation that, over time, the software development firm inevitably learns something about the fields of business of its clients, resulting in a transfer of some of the user's database to the manufacturer. Also, the

9. Rapid prototyping is a software development methodology in which the programmer interviews the user, gets some information about the application to be developed, and creates a prototype. The user then plays with the prototype and gives the manufacturer feedback, which causes the manufacturer to develop another, more refined prototype. This cycle continues until the user has a system that satisfies his requirements.

client learns something about the custom software development process, resulting in a transfer of some of the manufacturer's database to the user. I specifically looked for projects where this exchange of knowledge had not yet occurred.

The software company I studied was working on ten first-of-kind projects. I then categorized these ten on the basis of: (1) the client's expertise regarding the particular business problem he wanted the software to address, and (2) the client's understanding of custom software. I classified the clients as either "high" or "low" for both of these characteristics.

I determined how the projects were classified with respect to these two characteristics by interviewing the principals or project leaders of each of the ten first-of-kind clients and asking them questions about the business problem they wanted to automate. If the client described his business problem as "confusing (to him)" or said something like "I never had to work in the business office before, but I have to learn about it in order to fix this problem in my business," I classified his understanding of the business problem as low. Conversely, if the client clearly described the business problem he wanted to automate, or if people within the client organization stated things such as "the boss really understands how this organization runs," I classified his understanding of the business problem as high.

If the client could only articulate his need for software to "cut down on the work" or to "reduce personnel," I classified the firm as having a low understanding of custom software. If the client was already using software to analyze a component of his business problem, or if the client displayed some understanding of the software and the computer system being considered, I categorized his understanding of custom software as high.

After differentiating the ten first-of-kind projects on the basis of these two characteristics, I selected three to study over a two-year period: a venture capital firm (portfolio management), an insurance claims processing firm, and a group medical practice.

	Client Characteristics	
Project	Understanding of Problem To Be Automated	Understanding of Custom Software
Venture Capital	High	High
Insurance	High	Low
Medical	Low	Low

Table 1: Client Characteristics: Understanding of the Business Problem To Be Automated and Understanding Of Custom Software[10]

Once I selected the three projects to investigate, I examined a sample of a large set of requests for changes to the custom application software developed for each of these clients as well as all of the revisions issued by the software company to the client firm.

My data source was the software company's project notes. These notes itemized the source of each request and described each change requested. Where it was difficult to understand the actual change requested in these written records, I interviewed the programmers who worked on the projects and the people at each client site who dealt with the software development company during the life of the project.

---

10. In the sample of first-of-kind projects, I did not find an example of a project where the user's understanding of the problem to be automated was low while his or her understanding of custom software was high. Also, in the venture capital case, while I determined that the user's understanding of custom software was high, this project was the first time the user had actually been involved in a custom software development project. Hence, this was a first-of-kind project.

Each request for a change was an individual item recorded by a member of the software company (usually a programmer or the project manager). I identified a total of 516 requests affecting the three projects.

Each revision consisted of a set of specific changes to the existing functions of the application software and resulted from interactions between the user and the manufacturer (see Figure 1 for an example from the study). I had access to the 68 revisions issued by the firm. Table 2 shows the number of requests and the number of revisions for each of the three projects.

```
PORTFOLIO MANAGER
VERSION 1.65
01/12/89

New Features
------------
1) Income from securities.  A new trade type, "INCOME" has been added to represent income from interest
and dividends.  Transactions for these trades must have a 'delta' of 0, and the amount of income from
securities and total proceeds columns.

2) The extra company information has a new field, 'Lead VC'.  The portfolio summary report can be
filtered to include companies from one Lead VC only, or it can be printed as normal.

Enhancements
------------
1) The Cost Analysis report now has Total Value and Difference columns, in addition to the Total Cost
column.

2) The Trade Log reports (Investment, Divestiture, etc.) now summarize quarterly totals.

3) The Cost Analysis and Trade Log reports no longer print unnecessary zeroes on the right side of the
reports.

4) Multiple commitments for the same security at different prices are now handled properly.

Bugs Fixed
----------
1) The Trade Log reports did not always clear the rightmost columns.

2) A number of potential network bugs were found and corrected.
```

Figure 1: Sample of a Revision for the Venture Capital Project

I then sought to examine various qualities of the information transferred between the client firm and the software firm during the course of application development. To do this,

Project	Requests (n)	Revisions (n)
Venture Capital	102	22
Insurance	147	6
Medical	267	40
Total	516	68

Table 2: Number of Requests and Number of Revisions by Project

I focused on the 516 requests for changes since interactions between the manufacturer (the custom software firm) and the user (the client) during the life of each project had generated these requests.[11] For each request, I analyzed the initiator of the request (user or manufacturer), the granularity (chunks or items) of the messages transferred across the organizational boundaries, the completeness or incompleteness of the message transferred, and the content of users' requests (new features or bugs). The results are discussed below.

4.2 Findings

In this study, each of the three projects shared an identical objective: develop a custom software system to solve a specific user need. A typical initial statement of the user's need would be, "I need a portfolio management system to keep track of our venture capital funds."

---

11. Generally, the requests are functionally independent. However, because more than one person at the software company is recording the requests, this is not always true. For example, the client might request the following change: "Make the color of the data entry screens blue and the menu screens red." One programmer might record this as a single request that quotes the user as stated, while another programmer might record this as two requests that are: "(1) Make the color of the data entry screens blue, and (2) Make the color of the menu screens red."

At the beginning of each project, two rich, distinct databases existed: the client's understanding of his business problem and the software company's understanding of the custom software development process. Recall that all three projects were first-of-kind. The software company had no specific knowledge about the application they were about to develop before the project began and the client had not previously been involved in a custom software project. Each of the two databases (user and manufacturer) was initially in a separate organization and at a geographically distinct location.

Following are my findings for the three projects I studied with regard to the initiator of the request (user or manufacturer), the granularity (chunks vs. items) of the messages transferred across the organizational boundaries, the completeness or incompleteness of the messages transferred, and the content of user requests (new features vs. bugs).

4.2.1 Initiator of Request

Remember that my goal in this study was to understand how software development is partitioned between end user and software developer and how information is transferred -- or not transferred -- when two very different, rich databases are each held in different organizations. Consequently, it was important to determine whether the user or the manufacturer was initiating the requests for changes.

I used the written records of the firm to determine the initiator of each request. When the initiator of the request was unclear, I interviewed the programmer who recorded the request to determine the source of the request. The breakdown of user-initiated requests versus manufacturer-initiated requests is as follows:

Project	User-Initiated Requests (%)	Mfr-Initiated Requests (%)	Total (n)
Venture Capital	91	9	(102)
Insurance	89	11	(147)
Medical	86	14	(267)
Total	88	12	(516)

Table 3: Percentage of User-Initiated Requests
vs. Percentage of Manufacturer-Initiated Requests

Upon examining my database of requests and interviewing the programmers and the clients involved in each project, I found that most requests for changes came from the user. These results are consistent with the observation by von Hippel that the customer (user) is a significant source of new product ideas (von Hippel 1978).

Furthermore, a significant difference exists between the content of the user-initiated requests compared to the content of the manufacturer-initiated requests. The user-initiated requests were all a result of specific information or needs contained in the user environment. Conversely, in all cases except the medical system, the manufacturer-initiated requests were not based on an understanding of or familiarity with the user environment. Instead, they arose from an understanding of the manufacturer environment (such as the expertise required to achieve faster performance) or manufacturer-specific knowledge gained from other user projects (such as, "Here is a better way to look up an investment fund in the database").

4.2.2 Granularity of the Messages Transferred

I identified two levels of granularity -- a "chunk" and an "item" -- that can be distinguished on the basis of the amount of work involved for the software development firm in implementing the request. I define a "chunk" as a request for a new feature that usually represents a significant amount of software development. I define an "item" as a specific request that usually represents a change to an already implemented chunk.

For example, a request that would be classified as a chunk is: "Implement the investment report." This is a request that requires the programmer to perform a number of different tasks in order to implement the request. In this specific case, the programmer would have to write program code to perform several tasks: (1) format the investment report, (2) search through the investment database and perform the calculations necessary to generate the appropriate output, and (3) generate the appropriate output. In a language such as Clarion, this would require as many as one hundred lines of code that could take up to a day to write, test, and debug.

A request that would be classified as an item is: "Make the total column on the investment report two characters wider." In this case, the programmer only has to perform one task in order to implement the request -- that of widening a field on the report. In Clarion, this would require a change to one line of the program and would take fifteen minutes to write, test, and debug.

Table 4 shows the breakdown of chunks and items I observed in the sample. Interestingly, the vast majority of requests are for specific items. In all three projects, there was roughly the same percentage of chunks (20%) and items (80%). This indicates that the

Project	Chunks (%)	Items (%)	Total (n)
Venture Capital	22	78	(102)
Insurance	18	82	(147)
Medical	20	80	(267)
Total	20	80	(516)

Table 4: Granularity of Messages: Percentage of Chunks vs. Percentage of Items

granularity of the messages was independent of the two client characteristics: the client's understanding of the business problem to be automated and the client's understanding of custom software.

When we look at the breakdown of chunks and items by initiator of the request (user or manufacturer), we see that the proportion of user requests to manufacturer requests was independent of the size of the request. This is shown below in Table 5.

	Chunk (%)		Item (%)	
	User	Mfr	User	Mfr
Venture Capital	100	0	89	11
Insurance	81	19	91	9
Medical[12]	90	9	85	15
Total	90	10	87	13

Table 5: Percentage of Chunk Requests Initiated by the User and Manufacturer vs. Percentage of Item Requests

12. In the medical project, a representative from the manufacturer spent three weeks at the user site. During these three weeks, this representative worked as a consultant to understand the user's business problem (remember that the medical client had a low understanding of the problem to be automated and a low understanding of custom software). I considered all of the requests issued by this representative to be "user requests," since the representative from the manufacturer who traveled to the user site was not involved in actually implementing the system.

4.2.3 Completeness of the Messages Transferred

In each project, both user and manufacturer databases had to be drawn upon in order to first generate and then solve each change request. Von Hippel (1989, 1990) has suggested that there is a significant advantage to be gained by efficiently partitioning development tasks and by concentrating problem-solving in a single locus. Consequently, in this study I investigated whether (1) the rich data of each side, the software firm and the client who were in different loci, were in any substantial way transferred to the other side so that problem-solving could proceed in one locus, or (2) whether the data remained where they were and only requests for specific actions -- stripped of any rich database content -- moved between the two loci during the software development work, or (3) something "in-between" took place, e.g., the rich data were transferred from manufacturer to user in certain cases.

Evidence for the first pattern, the transfer of rich data between loci, would be interactions between the client and software firm in which (a) the client said, "Let us tell you all about our business so that you can develop a software package for us," or (b) "Teach us about how you write software so that we can figure out what we need from you," or (c) "Let's get together and learn about each other so that we can do this project jointly."

A series of discrete client-software firm interactions where the data remained where they were and only requests for specific actions moved between the two loci would be evidence for the second pattern. On the client side, we might expect to see extremely specific requests -- in effect, "Do this" -- devoid of user data regarding the context of the request or related reasoning. On the software firm side, we might expect to see very specific responses -- in effect, "We did this" -- also devoid of information regarding the context or any explanation of the reasoning used to develop the particular response.

Evidence for the third pattern, where something "in-between" took place, would be a combination of collaborative work between the user and manufacturer and specific requests from either the user or manufacturer. For example, the user and manufacturer might begin the project with a series of meetings to define the project. Then, the user would issue specific requests to the manufacturer during the life of the project. Finally, periodic "working sessions" would be held by the user and manufacturer to check the progress of the project and to reestablish priorities.

Interestingly, in all three projects I found that virtually all of the data passing back and forth between user and manufacturer were for specific requests -- essentially of a "Do this" nature -- thereby providing support for the second pattern noted above. I observed that the requests fell into two categories: (1) the initial "Do this" request was complete and unambiguous enough to allow the software firm to implement the request without further information; or (2) the "Do this" request needed further amplification before it could be implemented.

Typical unambiguous requests from the client (in this case, the venture capital company) would be: "The trade log does not always clear rightmost column -- it should" or, "Where you now print 'share' in the output reports, print 'shares' instead." The software company would make the requested change and issue another release of the software, effectively saying, "The software now does what you asked us for."

An example of a "Do this" request requiring further amplification in the venture capital project was, "Use percentage of cost method for revaluation." In the case where the "Do this" request required further amplification, the software company was only concerned with collecting the information needed to do what the client had requested -- not with evaluating the merit or validity or broader context of the request. In the above example,

the software company did not investigate cost valuation equations in order to see if there was a better solution to this particular request. Instead the programmer asked questions like, "How would you calculate the percentage of cost for a revaluation?" in order to understand how to implement the request.

I also found that a request for further amplification is much more likely for chunks than for items, as in Table 6.

Project	Do This Complete		Do This Incomplete		Total	
	Chk (%)	Itm (%)	Chk (%)	Itm (%)	Chk (n)	Itm (n)
Venture Capital	27	98	73	3	(22)	(80)
Insurance	37	98	63	2	(27)	(120)
Medical	47	99	53	1	(53)	(214)
Total	40	98	60	2	(102)	(414)

Table 6: Content of Messages: Percentage of "Do This Complete" Transferred vs. Percentage of "Do This Incomplete" Transferred

The request for further amplification was always initiated by the manufacturer. This happened because the manufacturer could not solve the problem without more information from the user. In cases where the client had a high understanding of the problem he wanted automated (in the venture capital and insurance firms), this led to a series of statements (not requests) by the user that told the manufacturer specifically how to achieve what the user had requested. In the medical example, where the client did not have a clear understanding of the problem he wanted to automate, the remote requests from the software firm for further amplifications of requests led to confusion, and the manufacturer eventually had to travel to the user site to resolve the confusion.

Hypothetically, it is possible for the manufacturer to learn the user's business and mix the two databases. But, according to the interviews I conducted, this is not what happens. The manufacturer obtains more information so that he can satisfy the "Do this" requests, but he does not necessarily understand the user's database.

In the case of the venture capital client, the users involved understood PC-based software and were often able to simulate what they wanted in terms of functions before making a "Do this" request of the software development firm. When the user is already familiar with some programming tools, she does a better job of framing the problem for the manufacturer. At the venture capital company, the user understood programming well enough to be able to prototype the problem in Lotus 1-2-3. Consequently, the user partitioned the problem and transmitted "Do this" requests that were clear and actionable by the remotely located manufacturer.

At the venture capital company, the users also understood their business well. In this case, their knowledge of the capability of the software firm was sufficient to allow effective interaction by both the user and the manufacturer at arm's length. Consequently, neither firm had to transfer any rich data to the other side to have an effective interaction regarding changes to the software.

In the case of the insurance claims processing company, the client understood his business reasonably well, but did not understand the software development process. The client also had a good manual system in place that needed to be automated. Consequently, the software firm studied the manual system in order to derive the series of "Do this" statements it needed to proceed. In this case, some rich data was transferred from the user to the manufacturer at the beginning of the project.

In the case of the medical practice, the client did not understand the benefit of custom software and could not transfer either specific requests or an understanding of its database to the remotely located software development firm. In addition, the client's understanding of its business problem was low. Specifically, the existing manual procedures the firm wanted to automate were poorly understood by the user. Therefore a representative of the software development firm ultimately had to travel to the client site and work there to effect this communication between user and manufacturer. While there, however, he did not collect a rich understanding of the user database and transfer it to the manufacturer. Instead, he concentrated on "understanding what we (the manufacturer) were supposed to do."

Finally, note that in none of the cases were software development tools transferred to the user site so that the user could modify the software provided by the manufacturer. In fact, it was the policy of the software company not to transfer such tools to the user so as to retain a proprietary advantage over other developers of custom software.

4.2.4 Content of User Requests: New Features vs. Bugs

I observed that 455 of the 516 requests were transmitted from the user site to the manufacturer site. Von Hippel (1989) has suggested that the traditional partitioning of tasks between the manufacturer and user, where users have needs and manufacturers have the task of assessing these needs and developing a responsive product, often raises significant problems for innovative projects because of the high interdependency of problem-solving. Consequently, in this study, I explored whether problems crossed back and forth across the boundary between the user and the manufacturer.

Because the rich user and manufacturer databases had been kept separate by the design of this study, the different types of requests by the user could in large part be further examined and categorized. I divided these requests into three categories: (1) new features, (2) bugs, and (3) specification errors.

New features were requests by the user for changes to the software program that resulted in the program performing new functions. These new features were typically either logical extensions of previous work that had been done or distinct new "modules" that automated processes that had not been previously automated. An example of a new feature request from the venture capital project is "Summarize quarterly totals on the trade logs." New features could either be a "chunk" or an "item." If a new feature required more than one programming task to implement, it was considered a chunk. If it required only one programming task, it was considered an item.

Bugs were functions that the manufacturer had implemented that did not work correctly. These are different from requests that were poorly defined by the user, classified below as "specification errors." In the case of bugs, the program simply did not work. An example of a bug from the venture capital project is "Transactions out of order cause the liquidation schedule to print the wrong information."

Specification errors were requests by the user that were poorly defined. In these cases, the user made a request that, when implemented by the manufacturer, prompted the user to revise the original request. While these requests were not actually bugs, they are clearly distinct from new features.

The classification of user requests into new features, bugs, and specification errors follows:

Project	New Features (%)	Bugs (%)	Specification Errors (%)	Total (n)
Venture Capital	41	31	28	(95)
Insurance	44	28	28	(131)
Medical	51	36	13	(229)
Total	47	33	20	(455)

Table 7: User Requests: Percentage of New Features vs. Percentage of Bugs vs. Percentage of Specification Errors

As shown in Table 7, roughly half (53%) of the requests from the client took the form of "Your software does not do what we asked." (This is the sum of the percentage of Bugs column and percentage of Specification Errors column.) These requests often traveled between the manufacturer and the user (across the boundary) twice and occasionally traveled across the boundary three times. This happened because the software development firm could not always adequately test the software without transferring it to the real-world client environment.

It is interesting to note that 20% of the user-initiated requests for changes shown in Table 7 were initiated because the user did not feel that the manufacturer had solved the initial request. This could happen either because the manufacturer did not solve the request correctly or because once the user saw how the manufacturer implemented the change, the user decided that he needed something slightly different.

In the case of the medical system, all of the specifications for new features came from the client or from a software development person working directly at the client site. In no case were the rich data needed to problem-solve and to generate such requests remotely transferred from the user site to the manufacturer site. Consequently, the programmers,

who were new to the topic area in each case, could not suggest a useful new function to the users on the basis of problem-solving carried out at the manufacturer site. This is consistent with the observation that most of the manufacturer suggestions are very technical in nature and relate specifically to the performance of the software.

4.3 Summary of Findings

In this pilot study, I examined various characteristics of the information transfer that took place between the end user and the software developer in three first-of-kind software application development projects. I found that in all three projects most of the data passing back and forth between user and manufacturer were for specific requests of a "Do this" nature. I observed that these requests fell into two categories: (1) the initial "Do this" request was complete and unambiguous enough to allow the software firm to implement the request without further information; or (2) the "Do this" request was incomplete and needed further amplification before it could be implemented. In addition, the request for further amplification was always initiated by the manufacturer.

I also found that most requests for changes came from the user (88%) rather than the manufacturer (12%). The user requests were all based on specific information contained within the user environment, while the manufacturer-initiated requests were based on manufacturer-specific knowledge.

Furthermore, I observed that roughly half (53%) of the requests from the user took the form of "Your software does not do what we asked" and often traveled between the manufacturer and the user (across the boundary) twice and occasionally traveled across the boundary three times. Almost all of the requests for new features came from the client or from a software development firm person working directly at the client site.

Finally, the granularity of the requests made varies -- there are requests for a significant amount of work (a chunk) and requests for a specific change (an item). However, I found that the percentage of chunks and items requested for a project is independent of the firm's understanding of its business problem and its understanding of custom software.

5 Trends in Software Technology

Von Hippel (1990) argues that there is an economy to be gained by concentrating problem-solving in one site rather than distributing it among two or more loci because data needed for problem-solving and innovation tend to be sticky. From this pilot study, I observed that the user is the source of most of the requests for new features in the software. I also observed that data are sticky and that problem-solving thus shifts back and forth between the two loci of sticky data; the user locus and the manufacturer locus. Simultaneously, there is a technology trend to make application software easier to develop (Ward 1986). If, as von Hippel suggests, innovation and problem-solving will end up in one locus, I predict that it will end up in the user locus. Programmers will continue to be busy; however, their jobs will be to create tools that enable the user to develop his or her application. The process of developing an application will become encapsulated in the tool. In other words, the tool will automate the traditional interaction between the programmer and the user. I suggest that this is a more optimal way to partition tasks between the user and the manufacturer than has previously been demonstrated.

5.1 An Example of User Empowerment

An example of this type of task partitioning is the development of expert systems within E.I. du Pont de Nemours & Company, Inc. The AI Technology group has taken a decentralized approach to expert systems development. The AI group played the role of the manufacturer, standardizing the tools and dispersing them to the user community (the entire organization). Users developed applications with the tools. These tools were continually evolving; however, users were shielded from the tool development by the manufacturer (the AI group), which monitored the tool evolution and provided users with the "latest and greatest" only when the "latest and greatest" worked (Computerworld 1987). The manufacturer (the AI group) controlled the development of the tools but not the development of the applications. Therefore, the value of the application to the user was not limited by the manufacturer's knowledge about the user's problem since the user developed the application.

In this example, the manufacturer concentrated on developing tools that allowed the user to develop his own application. The manufacturer did not have to transfer any of his database, which contained knowledge about developing the tools, to the user. Correspondingly, the user did not have to transfer any of his database, which contained knowledge about a wide variety of highly technical systems, to the manufacturer. The end result has been an extremely successful use of expert systems within Du Pont.

5.2 Custom Software vs. General Software

So far I have discussed custom application software -- software that is designed for a specific user's problem. I believe that the observations from this study are also applicable to more general software such as word processors, communication software, accounting

systems, and spreadsheets. I hypothesize that pieces of these applications will be built by programmers in such a way that the user can combine the pieces however he or she wants, with the extra (or, from the user's frame of reference, unnecessary) pieces being unobtrusive. Using the computer science metaphor of "object-oriented programming," word processors, communication software, accounting systems, and spreadsheets can be thought of as *objects* identical to the modules I described in the beginning of this paper. A user will be able to string these modules together to solve his or her specific application needs.

An example of a general software system using generic objects that a user can configure to meet his specific needs is Microstep.[13] An example of the implementation of a specific system using Microstep is as follows: "If a programmer wants to add a payroll function for the sales department, and he or she has already built such a routine for the marketing department, rewriting it for sales is as simple as pointing at a symbol for payroll, dragging it to the marketing design, and connecting it to the rest of the marketing design with a line. [It was] done in less than two minutes, and that included pauses for explanations" (Lewis 1988).

These generic objects will still be developed by software engineers. Users will not develop the object-oriented programming and the CASE tools of the future, nor will they develop the generic modules that one can consider to be part of the tool. Instead, the role of the manufacturer will be to evolve tools such as these to a higher degree of functionality. The user will then bring them into his or her environment and configure them to his or her specific needs.

---

13. Syscorp International.

5.3 Disintermediation: A Possible Consequence

If users generate the majority of application programs, the commercial providers of application programs may begin to disappear. This will resemble the disintermediation that has occurred in other industries (e.g., the direct placement of airline reservations, or the use of on-line computer services such as Compuserve instead of retail brokerages to buy and sell stocks).

This disintermediation has already occurred in discrete segments of the computer software market. For example, when Lotus 1-2-3 first appeared on the market, commercial software providers rushed to develop "1-2-3 Templates." Initially, it appeared that a respectable template market would exist. However, as users discovered how easy it was to develop their own custom templates, the template market disappeared.

Currently, there is a large demand for decentralized application software. Numerous firms exist to provide custom software solutions to all sizes of users from small businesses to Fortune 500 companies. Most of these companies approach application development in a very labor-intensive manner. Many functional experts are hired to specify and interpret user information and transfer it across the boundary to the technical experts in the manufacturer organization, who then develop the software.

I would predict that the core business of these software companies will eventually be jeopardized. Instead of needing a massive functional organization to interpret user needs and design applications systems around these user needs, these software companies will need to be able to transfer their process technology to the user environment and then teach the user how to use the tools to develop application software. The manufacturer firms will

not add value in the development of application software for a specific user need. Instead, they will add value by developing and transferring tools into the user environment at the appropriate places.

The need for specialist programmers will not be restricted to just the development of tools even if disintermediation occurs. For example, large organizations will continue to require the services of programmers to maintain formal structures for standardizing information systems across functional and organizational boundaries.

Consider Frito-Lay, Inc., a leading manufacturer and distributor of snack-foods. In the 1970's and early 1980's, Frito-Lay had a traditional, centralized Management Information Systems (MIS) department. However, since 1985, Frito-Lay systematically decentralized the gathering of information (through the use of hand-held computers by the route drivers) and the process of making decisions that occurs in the organization (through the use of an Executive Information System on managers' desks). A centralized "data warehouse," containing all of the transactional data for the organization, is maintained by the MIS department. However, the data are no longer analyzed by the MIS department, as they once were. Instead, data are dispersed to the managers (in many cases as frequently as once a day) in order to be used for analysis, production planning, sales strategies, and promotion strategies. The MIS department no longer writes application software to perform specific, user-requested analysis on the data. Instead, it develops the tools used for the analysis. The users responsible for making the decisions perform the actual analysis using the tools developed by the MIS department. Ultimately, this system will provide information to the decision makers across organization boundaries on a timely basis. The users will be able to tailor specific applications to their needs. The manufacturer (MIS) will provide the necessary tools and standards for collecting and storing data (Applegate

1989).

In the du Pont example, the AI group recentralized the applications when it was appropriate. They did not take responsibility for maintaining and evolving the application -- the user that developed the application was still responsible for it. However, the AI group helped disperse generally useful tools to other parts of the organization. Also, the AI group helped develop the links to other, older systems that would otherwise be inaccessible to the end users.

In both of these cases, there is a need for some programming to be done by the manufacturer. However, this programming is driven by the need for consistency across multiple user boundaries. I believe that in these cases, tasks would be most effectively partitioned by users developing their applications around a data framework designed and maintained by the manufacturer.

6 Conclusion

The role of the user in the software development process has changed dramatically over the last few decades. In this paper, I undertook a pilot study to establish an empirical basis for research on the actual partitioning of application programming tasks between user and manufacturer. While the results of the present study do not pinpoint the optimal way to partition tasks between the end user and specialist programmer of application software, they do demonstrate that both the user and manufacturer have rich databases of information which are not easily transferred across the user-manufacturer boundary. Therefore, it is important to consider task partitioning and the implications of sticky data for the locus of innovation and problem-solving in the software industry. Finally, they suggest the value of the manufacturer developing tools based on expertise in the

manufacturer's environment and then transferring these tools to the user's environment so the user can develop his or her application independently of the manufacturer. I believe the issues explored in this paper create a substantial basis for a more rigorous empirical study of different approaches to task partitioning and the impact of sticky data in application software projects.

References

Applegate, Linda. 1989. "Frito-Lay, Inc., A Strategic Transition (C)." Harvard Business School Case Study #9-190-071, Harvard Business School, Boston, Mass., December 1989.

"Artificial Intelligence Spotlight: Interview with Ed Mahler -- Dupont." *Computerworld*, (November 1987): 59.

Henderson, John, and Soonchul Lee. 1989. "I/S Design Team Performance: A Control Theory Perspective." Sloan School of Management, Massachusetts Institute of Technology, Cambridge, Mass.

Lewis, Peter H. 1988. "The Executive Computer." *The New York Times*, 26 February 1988, F11.

Meng, Brita. 1990. "Object-Oriented Programming." *Macworld* (January 1990): 174-80.

Orlikowski, Wanda J. 1989. "Division Among the Ranks: The Social Implications of CASE Tools for System Developers," 199-210. In Proceedings of the Tenth International Conference on Information Systems, 4-6 December 1989, Boston.

Salaway, G. 1987. "An Organizational Learning Approach to Information Systems Development." *MIS Quarterly* 11, no. 2 (1987): 245-64.

von Hippel, Eric. 1978. "Successful Industrial Products From Customer Ideas." *Journal of Marketing* 42, no. 1 (January 1978): 39-49.

von Hippel, Eric. 1990. "The Impact of 'Sticky Data' on Innovation and Problem-Solving." Working Paper No. 3147-90-BPS, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, Mass., April 1990.

von Hippel, Eric. 1989. "Shifting Product and Service Prototyping to Users: An Innovation Process Advantage?" Working Paper No. 3032-89-BPS, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, Mass., June 1989.

von Hippel, Eric. 1988. *The Sources of Innovation*. New York: Oxford University Press.

von Hippel, Eric. 1989. "Task Partitioning: An Innovation Process Variable." Working Paper No. 2030-88, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, Mass., April 1989.

Ward, Paul T. 1986. *System Development Without Pain: A User's Guide to Modeling Organizational Patterns*. New York: Yourdon Press.

Wegner, Peter. 1989. "Learning the Language." *Byte Magazine*, 14 (March 1989): 245-50.

1926 101

Date Due

JUN 1992 1991

MAR. 2 4 1995

Lib-26-67

MIT LIBRARIES DUPL
3 9080 00699020 1